# Министерство Образования Республики Беларусь

Учреждение образования

«Гомельский государственный технический университет

имени П.О. Сухого»

Кафедра «Информатика»

по курсу: «Разработка приложений для Интернет»

Лабораторная работа №5

# «Работа с базами данных в Node.js»

Выполнил: студент группы ИП-31

Коваленко Анастасия Игоревна

Допуск к защите: Проверил: преподаватель

Дата защиты: Самовендюк Николай Владимирович

Гомель 2023

**Цель**: изучить основы работы по написанию скриптов для работы с базами данных в Node.js. Получить навыки работы с orm в Node.js. Научиться создавать и тестировать простые приложения.

**Практическая часть:**

**Задание 1** – Реализовать Rest Api (добавлении, удаление, обновление, получение) с использование express и базы данных. Предметную область взять по усмотрению студента. Использовать sql запросы к обращению к базе данных.

**Листинг программы:**

const express = require("express");

const sqlite3 = require("sqlite3");

const isTest = process.argv.slice(-1)[0] === "--test";

const app = express();

const db = new sqlite3.Database("./database/movie.db", (err) => {

if (err) {

console.error(err.message);

return;

}

if (!isTest) {

console.log("Successfully connected to the database...");

}

});

function isMovieValid(movie){

return movie && Number.isInteger(movie.id) && movie.id >= 0 && movie.name !== undefined

&& movie.description !== undefined && Number.isInteger(movie.price) && movie.price >= 0 &&

movie.genre !== undefined;

}

function getObjectFromRow(row) {

return {

id: row.Id,

name: row.Name,

price: row.Price,

description: row.Description,

genre: row.Genre

};

}

app.use(express.json());

app

.route("/movie")

.get((request, response) => {

const id = request.query.id;

if(id) {

db.serialize(function() {

db.get("SELECT \* FROM Movie WHERE Id = ?;", id, (err, row) =>{

if(err){

response.status(500).send(err.message);

return;

}

if (row === undefined) {

response.status(404).send(`Movie with 'id' = ${id} not found.`);

return;

}

response.status(200).json({ movie: getObjectFromRow(row) });

})

})

}

else{

db.serialize(function() {

db.all("SELECT \* FROM Movie;", (err, rows) => {

if (err) {

response.status(500).send(err.message);

return;

}

let array = [];

rows.forEach((row) => array.push(getObjectFromRow(row)));

response.status(200).json({ movies: array });

});

});

}

})

.post((request, response) =>{

const movie = request.body.movie;

if(isMovieValid(movie)){

db.serialize(function() {

db.get("SELECT COUNT(\*) AS 'Count' FROM Movie WHERE Id = ?;", movie.id, (err, row) => {

if (err) {

response.status(500).send(err.message);

return;

}

if (row.Count != 0) {

response.status(400).send(`Movie with 'id' = ${movie.id} is already exists in the database.`);

return;

}

db.run("INSERT INTO Movie VALUES(?, ?, ?, ?, ?);", [ movie.id, movie.name, movie.price, movie.description, movie.genre ], (err) => {

if (err) {

response.status(500).send(err.message);

return;

}

response.status(200).send(`Movie with 'id' = ${movie.id} was successfully added to the database.`);

});

});

});

}

else{

response.status(400).send("Bad request body: 'movie' not present correctly.");

}

})

.put((request, response) =>{

const movie = request.body.movie;

if(isMovieValid(movie)){

db.serialize(function() {

db.get("SELECT COUNT(\*) AS 'Count' FROM Movie WHERE Id = ?;", movie.id, (err, row) => {

if (err) {

response.status(500).send(err.message);

return;

}

if (row.Count == 0) {

response.status(404).send(`Movie with 'id' = ${movie.id} is not found in the database.`);

return;

}

db.run("UPDATE Movie SET Name = ?, Price = ?, Description = ?, Genre = ? WHERE Id = ?;",

[ movie.name, movie.price, movie.description, movie.genre, movie.id ], (err) => {

if (err) {

response.status(500).send(err.message);

return;

}

response.status(200).send(`Movie with 'id' = ${movie.id} was successfully updated.`);

});

});

});

}

else{

response.status(400).send("Bad request body: 'movie' not present correctly.");

}

})

.delete((request, response) =>{

const id = request.query.id;

if(id){

db.serialize(function() {

db.get("SELECT COUNT(\*) AS 'Count' FROM Movie WHERE Id = ?;", id, (err, row) => {

if (err) {

response.status(500).send(err.message);

return;

}

if (row.Count == 0) {

response.status(404).send(`Movie with 'id' = ${id} is not found in the database.`);

return;

}

db.run("DELETE FROM Movie WHERE Id = ?;", id, (err) => {

if (err) {

response.status(500).send(err.message);

return;

}

response.status(200).send(`Movie with 'id' = ${id} was successfully deleted.`);

});

});

});

}

else{

response.status(400).send("Bad request body: 'request must contains 'id' parameter.");

}

});

if(!isTest){

app.listen(3000, () => console.log("Server is listening on port 3000"));

}

else{

module.exports.app = app; }

**Задание 2** – Повторить задание 1, но с использование ORM.

**Листинг программы:**

const express = require("express");

const { Sequelize, DataTypes } = require('sequelize');

const isTest = process.argv.slice(-1)[0] === "--test";

const app = express();

const sequelize = new Sequelize('sqlite:./database/movie.db', { logging: !isTest });

const Movie = sequelize.define('Movie', {

Id: {

type: DataTypes.INTEGER,

autoIncrement: true,

primaryKey: true,

allowNull: false

},

Name:{

type: DataTypes.TEXT,

allowNull: false

},

Price:{

type: DataTypes.INTEGER,

allowNull: false

},

Description:{

type: DataTypes.TEXT,

allowNull: false

},

Genre:{

type: DataTypes.TEXT,

allowNull: false

}

},{

timestamps: false

})

function isMovieValid(movie){

return movie && Number.isInteger(movie.id) && movie.id >= 0 && movie.name !== undefined

&& movie.description !== undefined && Number.isInteger(movie.price) && movie.price >= 0 &&

movie.genre !== undefined;

}

function getObjectFromModel(model) {

return {

id: model.Id,

name: model.Name,

price: model.Price,

description: model.Description,

genre: model.Genre

};

}

app.use(express.json());

app

.route("/movie")

.get((request, response) => {

const id = request.query.id;

if(id) {

Movie.findOne({where: {Id: id}})

.then((model) =>{

if(model === null){

response.status(404).send(`Movie with 'id' = ${id} not found.`);

return;

}

response.status(200).json({ movie: getObjectFromModel(model) });

})

.catch((err) => response.status(500).send(err.message));

}

else{

Movie.findAll()

.then((models) =>{

let array = [];

models.forEach((model) => array.push(getObjectFromModel(model)));

response.status(200).json({ movies: array });

})

.catch((err) => response.status(500).send(err.message));

}

})

.post((request, response) =>{

const movie = request.body.movie;

if(isMovieValid(movie)){

Movie.findOrCreate({

where: {Id: movie.id},

defaults:{

Id: model.id,

Name: model.name,

Price: model.price,

Description: model.description,

Genre: model.genre

}

})

.then((result) => {

if (result[1] === false) {

response.status(400).send(`Movie with 'id' = ${movie.id} already exists in the database.`);

return;

}

response.status(200).send(`Movie with 'id' = ${movie.id} was successfully added to the database.`);

})

.catch((err) => response.status(500).send(err.message));

}

else{

response.status(400).send("Bad request body: 'movie' not present correctly.");

}

})

.put((request, response) =>{

const movie = request.body.movie;

if(isMovieValid(movie)){

Movie.update({

Name: model.name,

Price: model.price,

Description: model.description,

Genre: model.genre

},{

where: {

Id: movie.id

}

})

.then((count) => {

if (count == 0) {

response.status(404).send(`Movie with 'id' = ${movie.id} not found in the database.`);

return;

}

response.status(200).send(`Movie with 'id' = ${movie.id} was successfully updated.`);

})

.catch((err) => response.status(500).send(err.message));

}

else{

response.status(400).send("Bad request body: 'movie' not present correctly.");

}

})

.delete((request, response) =>{

const id = request.query.id;

if(id){

Movie.destroy({where: {Id : id}})

.then(count => {

if(count ==0){

response.status(404).send(`Movie with 'id' = ${movie.id} not found in the database.`);

return;

}

response.status(200).send(`Movie with 'id' = ${movie.id} was successfully deleted.`);

})

.catch((err) => response.status(500).send(err.message));

}

else{

response.status(400).send("Bad delete request body: 'must contains 'id' parameter.");

}

});

sequelize.sync()

.then(() =>{

if(!isTest){

console.log("Successfully connected to the database...");

app.listen(3000, () => console.log("Server is listening on port 3000..."));

}

})

.catch((err) => console.error(err));

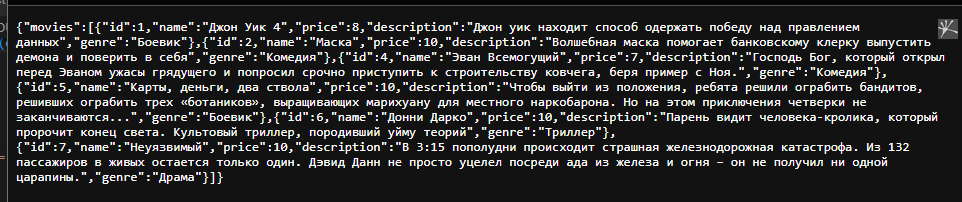
if(!isTest){

module.exports.app = app;

}

**Результат выполнение программы**

На рисунке 1 изображен результат выполнения задания 1



**Листинг тестов:**

**test5\_1.js**

const request = require("supertest");

const assert = require("assert");

const app = require("../lab5\_1/app").app;

const movie3 ={

movie:{

id:3,

name: "Gercules",

price: 8,

description: "greek mythology",

genre: "cartoon"

}

};

const movie3\_upd ={

movie:{

id:3,

name: "Cars",

price: 8,

description: "about race",

genre: "cartoon"

}

};

describe("Testing API with SQL queries", () => {

describe("GET /movie", () => {

it("Should return 200 \"OK\" with JSON", (done) => {

request(app)

.get("/movie")

.expect("Content-Type", /json/)

.expect(200)

.end(done);

});

});

describe("GET /movie?id=3 NON-EXISTENT", () => {

it("Should return 404 \"Not Found\" with Text", (done) => {

request(app)

.get("/movie")

.query({id: 0})

.expect("Content-Type", /text/)

.expect(404)

.end(done);

});

});

describe("POST /movie with id=3 NON-EXISTENT", () => {

it("Should return 200 \"OK\" with Text", (done) => {

request(app)

.post("/movie")

.send(movie3)

.expect("Content-Type", /text/)

.expect(200)

.end(done);

});

});

describe("GET /movie?id=3 ADDED", () => {

it("Should return 200 \"OK\" with expected JSON", (done) => {

request(app)

.get("/movie")

.query({id: 3})

.expect("Content-Type", /json/)

.expect(200)

.expect((response) => {

assert.deepStrictEqual(response.body, movie3);

})

.end(done);

});

});

describe("POST /movie with id=3 EXISTENT", () => {

it("Should return 400 \"Bad Request\" with Text", (done) => {

request(app)

.post("/movie")

.send(movie3)

.expect("Content-Type", /text/)

.expect(400)

.end(done);

});

});

describe("PUT /movie with id=3 EXISTENT", () => {

it("Should return 200 \"OK\" with Text", (done) => {

request(app)

.put("/movie")

.send(movie3\_upd)

.expect("Content-Type", /text/)

.expect(200)

.end(done);

});

});

describe("GET /movie?id=3 UPDATED", () => {

it("Should return 200 \"OK\" with expected JSON", (done) => {

request(app)

.get("/movie")

.query({id: 3})

.expect("Content-Type", /json/)

.expect(200)

.expect((response) => {

assert.deepStrictEqual(response.body, movie3\_upd);

})

.end(done);

});

});

describe("DELETE /movie with id=3 EXISTENT", () => {

it("Should return 200 \"OK\" with Text", (done) => {

request(app)

.delete("/movie")

.query({id: 3})

.expect("Content-Type", /text/)

.expect(200)

.end(done);

});

});

describe("GET /movie?id=3 DELETED", () => {

it("Should return 404 \"Not Found\" with Text", (done) => {

request(app)

.get("/movie")

.query({id: 0})

.expect("Content-Type", /text/)

.expect(404)

.end(done);

});

});

describe("PUT /movie with id=3 DELETED", () => {

it("Should return 404 \"Not Found\" with Text", (done) => {

request(app)

.put("/movie")

.send(movie3\_upd)

.expect("Content-Type", /text/)

.expect(404)

.end(done);

});

});

describe("DELETE /movie with id=3 DELETED", () => {

it("Should return 404 \"Not Found\" with Text", (done) => {

request(app)

.delete("/movie")

.query({id: 3})

.expect("Content-Type", /text/)

.expect(404)

.end(done);

});

});

});

**test5\_2.js**

const request = require("supertest");

const assert = require("assert");

const app = require("../lab5\_1/app").app;

const movie3 ={

movie:{

id:3,

name: "Gercules",

price: 8,

description: "greek mythology",

genre: "cartoon"

}

};

const movie3\_upd ={

movie:{

id:3,

name: "Cars",

price: 8,

description: "about race",

genre: "cartoon"

}

};

describe("Testing API with ORM Sequelize", () => {

describe("GET /movie", () => {

it("Should return 200 \"OK\" with JSON", (done) => {

request(app)

.get("/movie")

.expect("Content-Type", /json/)

.expect(200)

.end(done);

});

});

describe("GET /movie?id=3 NON-EXISTENT", () => {

it("Should return 404 \"Not Found\" with Text", (done) => {

request(app)

.get("/movie")

.query({id: 0})

.expect("Content-Type", /text/)

.expect(404)

.end(done);

});

});

describe("POST /movie with id=3 NON-EXISTENT", () => {

it("Should return 200 \"OK\" with Text", (done) => {

request(app)

.post("/movie")

.send(movie3)

.expect("Content-Type", /text/)

.expect(200)

.end(done);

});

});

describe("GET /movie?id=3 ADDED", () => {

it("Should return 200 \"OK\" with expected JSON", (done) => {

request(app)

.get("/movie")

.query({id: 3})

.expect("Content-Type", /json/)

.expect(200)

.expect((response) => {

assert.deepStrictEqual(response.body, movie3);

})

.end(done);

});

});

describe("POST /movie with id=3 EXISTENT", () => {

it("Should return 400 \"Bad Request\" with Text", (done) => {

request(app)

.post("/movie")

.send(movie3)

.expect("Content-Type", /text/)

.expect(400)

.end(done);

});

});

describe("PUT /movie with id=3 EXISTENT", () => {

it("Should return 200 \"OK\" with Text", (done) => {

request(app)

.put("/movie")

.send(movie3\_upd)

.expect("Content-Type", /text/)

.expect(200)

.end(done);

});

});

describe("GET /movie?id=3 UPDATED", () => {

it("Should return 200 \"OK\" with expected JSON", (done) => {

request(app)

.get("/movie")

.query({id: 3})

.expect("Content-Type", /json/)

.expect(200)

.expect((response) => {

assert.deepStrictEqual(response.body, movie3\_upd);

})

.end(done);

});

});

describe("DELETE /movie with id=3 EXISTENT", () => {

it("Should return 200 \"OK\" with Text", (done) => {

request(app)

.delete("/movie")

.query({id: 3})

.expect("Content-Type", /text/)

.expect(200)

.end(done);

});

});

describe("GET /movie?id=3 DELETED", () => {

it("Should return 404 \"Not Found\" with Text", (done) => {

request(app)

.get("/movie")

.query({id: 0})

.expect("Content-Type", /text/)

.expect(404)

.end(done);

});

});

describe("PUT /movie with id=3 DELETED", () => {

it("Should return 404 \"Not Found\" with Text", (done) => {

request(app)

.put("/movie")

.send(movie3\_upd)

.expect("Content-Type", /text/)

.expect(404)

.end(done);

});

});

describe("DELETE /movie with id=3 DELETED", () => {

it("Should return 404 \"Not Found\" with Text", (done) => {

request(app)

.delete("/movie")

.query({id: 3})

.expect("Content-Type", /text/)

.expect(404)

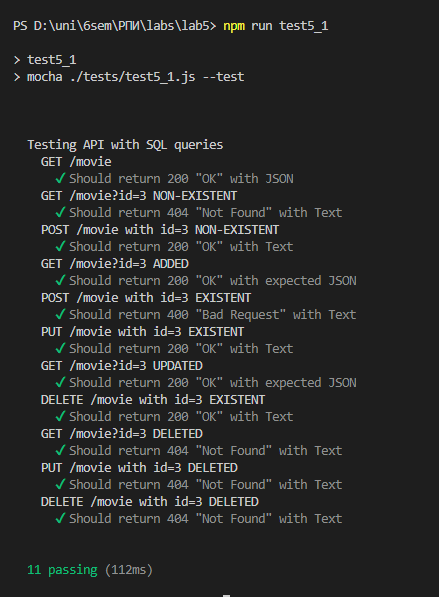
.end(done);

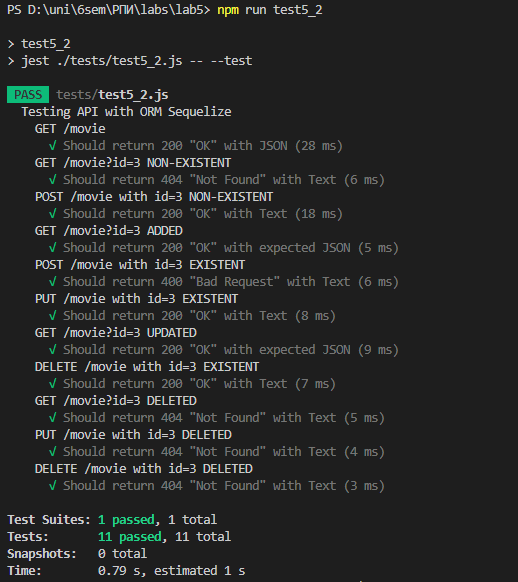
});

});

});

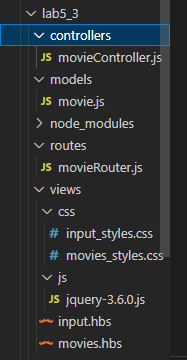
**Результаты тестов:**

****

****

**Задание 4** – Реализовать паттерн MVC, создать три папки в которых будут хранится классы моделей, контроллеров и представлений. Для представлений можно использовать handlebars. Классы сервисов и маршрутов находятся в отдельных файлах.

**Структура проекта:**

****

**movieController.js**

const { is } = require("express/lib/request");

const { Movie } = require("../models/movie");

function isMovieValid(movie){

let id = Number(movie.id);

let price = Number(movie.price);

console.error(id)

console.error(Number.isInteger(id))

console.error(isNaN(price))

movie.id = id;

movie.price = price;

return movie && Number.isInteger(movie.id) && movie.id >= 0 && movie.name !== undefined

&& movie.description !== undefined && Number.isInteger(movie.price) && movie.price >= 0 &&

movie.genre !== undefined;

}

function getObjectFromModel(model) {

return {

id: model.Id,

name: model.Name,

price: model.Price,

description: model.Description,

genre: model.Genre

};

}

module.exports.getAllMovies = (\_, response) => {

Movie.findAll()

.then((models) => {

let array = [];

models.forEach((model) => array.push(getObjectFromModel(model)));

response.render("movies.hbs", {

movies: array

});

})

.catch((err) => response.status(500).send(err));

};

module.exports.getMovieInput = (request, response) => {

const action = request.query.action;

if (action && (action === "create" || action === "update")) {

if (action === "create") {

response.render("input.hbs", {

path: "create"

});

}

else if (request.query.id) {

Movie.findOne({ where: { Id: request.query.id }})

.then((model) => {

if (model === null) {

response.status(404).send(`Movie with 'id' = ${id} not found.`);

return;

}

response.render("input.hbs", {

path: "update",

tour: getObjectFromModel(model)

});

})

.catch((err) => response.status(500).send(err.message));

}

else {

response.status(400).send("Bad request: must contains 'id' parameter.");

}

}

else {

response.status(400).send("Bad request: must contains 'action' parameter.");

}

};

module.exports.createMovie = (request, response) => {

const body = request.body;

if (isMovieValid(body)) {

Movie.findOrCreate({

where: {Id: body.id},

defaults:{

Id: body.id,

Name: body.name,

Price: body.price,

Description: body.description,

Genre: body.genre

}

})

.then((result) => {

if (result[1] === false) {

response.status(400).send(`Movie with 'id' = ${body.id} already exists in the database.`);

return;

}

response.redirect("/movie");

})

.catch((err) => response.status(500).send(err.message));

}

else {

response.status(400).send("Bad request body: parameters not present correctly.");

}

};

module.exports.updateMovie = (request, response) => {

const body = request.body;

if (isMovieValid(body)) {

Movie.update({

Name: body.name,

Price: body.price,

Description: body.description,

Genre: body.genre

},{

where: {

Id: body.id

}

})

.then(count => {

if (count == 0) {

response.status(404).send(`Movie with 'id' = ${body.id} not found.`);

return;

}

response.redirect("/movie");

})

.catch((err) => response.status(500).send(err.message));

}

else {

response.status(400).send("Bad request body: parameters not present correctly.");

}

};

module.exports.deleteMovie = (request, response) => {

const id = request.body.id;

if (id) {

Movie.destroy({ where: { Id: id } })

.then(count => {

if (count == 0) {

response.status(404).send(`Movie with 'id' = ${id} not found.`);

return;

}

response.redirect("/movie");

})

.catch((err) => response.status(500).send(err.message));

}

else {

response.status(400).send("Bad request body: must contains 'id' parameter.");

}

};

**movie.js**

const { DataTypes } = require('sequelize');

const { sequelize } = require("../app");

const Movie = sequelize.define('Movie', {

Id: {

type: DataTypes.INTEGER,

autoIncrement: true,

primaryKey: true,

allowNull: false

},

Name:{

type: DataTypes.TEXT,

allowNull: false

},

Price:{

type: DataTypes.INTEGER,

allowNull: false

},

Description:{

type: DataTypes.TEXT,

allowNull: false

},

Genre:{

type: DataTypes.TEXT,

allowNull: false

}

},{

timestamps: false

});

module.exports.Movie = Movie;

**movieRouter.js**

const express = require("express");

const movieController = require("../controllers/movieController");

const movieRouter = express.Router();

movieRouter.get("/", movieController.getAllMovies);

movieRouter.get("/input", movieController.getMovieInput);

movieRouter.post("/create", movieController.createMovie);

movieRouter.post("/update", movieController.updateMovie);

movieRouter.post("/delete", movieController.deleteMovie);

module.exports.movieRouter = movieRouter;

**input.hbs**

<html lang="en">

<head>

<title>Ввод фильма</title>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="stylesheet" href="/css/input\_styles.css">

</head>

<body>

<h2>Ввод фильма</h2>

<form action="/movie/{{path}}" method="POST">

<label>ID:</label><br>

<input type="text" name="id" value="{{movie.id}}"><br><br>

<label>Название:</label><br>

<input type="text" name="name" value="{{movie.name}}"><br><br>

<label>Цена:</label><br>

<input type="text" name="price" value="{{movie.price}}"><br><br>

<label>Описание:</label><br>

<input type="text" name="description" value="{{movie.description}}"><br><br>

<label>Жанр:</label><br>

<input type="text" name="genre" value="{{movie.genre}}"><br><br>

<input type="submit" value="OK">

</form>

</body>

</html>

**movies.hbs**

<!DOCTYPE html>

<html>

<head>

<title>Фильмы</title>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="stylesheet" href="/css/movies\_styles.css">

<script type="text/javascript" src="/js/jquery-3.6.0.js"></script>

</head>

<body>

<h2>Таблица фильмов</h2>

<table>

<tr>

<th>ID</th>

<th>Название</th>

<th>Цена</th>

<th>Описание</th>

<th>Жанр</th>

</tr>

{{#each movies}}

<tr>

<td>{{this.id}}</td>

<td>{{this.name}}</td>

<td>{{this.price}}</td>

<td>{{this.description}}</td>

<td>{{this.genre}}</td>

</tr>

{{/each}}

</table>

<div style="margin-top: 10px;">

<form id="inputForm" action="/movie/input" mathod="GET">

<input type="hidden" id="inputFormAction" name="action">

<input type="hidden" id="inputIdField" name="id">

<input type="submit" id="createButton" value="Добавить фильм">

<input type="submit" id="updateButton" value="Изменить фильм">

</form>

<form id="deleteForm" action="/movie/delete" method="POST">

<input type="hidden" id="deleteIdField" name="id">

<input type="submit" id="delete" value="Удалить фильм">

</form>

</div>

<script>

let lastSelected = -1;

$("tr").click((e) => {

let nextSelected = $("tr").index(e.currentTarget);

if (lastSelected != -1 && nextSelected != 0) {

$($("tr")[lastSelected]).css("background-color", "");

}

if (nextSelected != 0) {

$(e.currentTarget).css("background-color", "limegreen");

lastSelected = nextSelected;

}

});

$("#inputForm").submit((e) => {

if ($(e.originalEvent.submitter).is($("#createButton"))) {

$("#inputFormAction").val("create");

}

else {

if (lastSelected == -1) {

e.preventDefault();

}

else {

$("#inputFormAction").val("update");

$("#inputIdField").val($($("tr")[lastSelected]).children().first().text());

}

}

});

$("#deleteForm").submit((e) => {

if (lastSelected == -1) {

e.preventDefault();

}

else {

$("#deleteIdField").val($($("tr")[lastSelected]).children().first().text());

}

});

</script>

</body>

<html>

**input\_styles.css**

form {

display: inline-block;

}

input[type=text] {

position: sticky;

}

input[type=submit] {

width: 100%;

}

**movies\_styles.css**

th, td, table {

border: 2px solid black;

}

th, td {

padding: 5px;

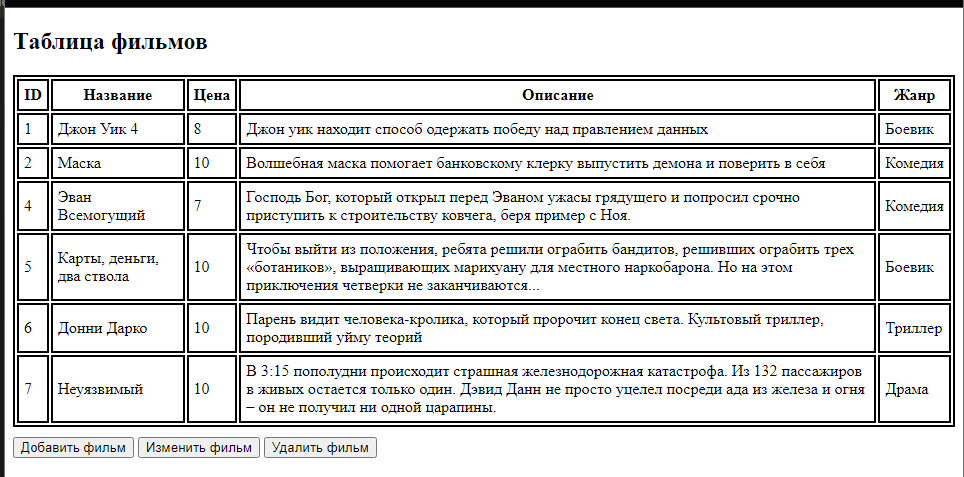
}

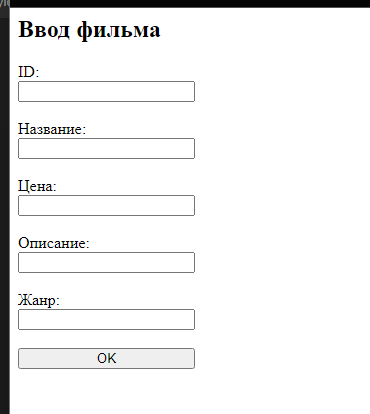
form {

display: inline-block;

}

**Результат выполнения:**

****

****

**Вывод:** в процессе выполнения лабораторной работы были изучены основы работы по написанию скриптов для работы с базами данных в Node.js, получены навыки работы с orm в Node.js. Научились создавать и тестировать простые приложения.